Notes on 100 Gbps (ongoing)

Brett Viren

October 4, 2019

1 Overview

This is a set of notes while understanding the ATLAS/DUNE FELIX DAQ test machines with high-speed 100 Gbps networking. The notes are approximately in order changes made to the system.

1.1 SSH

The FELIX DAQ computers are down in the ATLAS lab. They are accessed on a private network via a router "felix.phy.bnl.gov" with SSH poking out to the BNL internal network on select ports:

3422 `sui`

3522 `dune`

After making my account and populating SSH keys I set up `~.ssh/config` so eg `ssh dune.felix` gets me in:

```
host dune.felix
     User bviren
     Hostname felix.phy.bnl.gov
     Port 3522
     ForwardAgent yes

host sui.felix
     User bviren
     Hostname felix.phy.bnl.gov
     Port 3422
     ForwardAgent yes
```

I make my account:

```
# useradd -Gwheel -c "Brett Viren" -m -s /bin/bash bviren
# passwd bviren
```

The `-Gwheel` gives `sudo` access but in order to make it easier to gain `root` I also add my SSH public key to `/root/.ssh/authorized_keys` and modify `/etc/ssh/sshd_config` to have:

```
PermitRootLogin prohibit-password
```

After restarting the SSH daemon this lets `ssh root@dune.felix` work with SSH keys but still denies use of a root password.

1.2 Systems

1.2.1 dune

Bits from dune's dmesg:

```
ixgbe: Intel(R) 10 Gigabit PCI Express Network Driver - version 5.1.0-k-rh7.6
i40e: Intel(R) Ethernet Connection XL710 Network Driver - version 2.3.2-k
mlx5_core 0000:18:00.1: MLX5E: StrdRq(1) RqSz(8) StrdSz(64) RxCqeCmprss(0)
mlx5_ib: Mellanox Connect-IB Infiniband driver v5.0-0
mlx5_core 0000:18:00.1 enp24s0f1: Link up
```

And ifconfig

```
enp24s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::b380:419d:570f:24c9  prefixlen 64  scopeid 0x20<link>
        ether 98:03:9b:97:40:2f  txqueuelen 1000  (Ethernet)
        RX packets 104383  bytes 34995092 (33.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52444  bytes 8739224 (8.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

And lspci

```
18:00.0 Ethernet controller: Mellanox Technologies MT27800 Family [ConnectX-5]
18:00.1 Ethernet controller: Mellanox Technologies MT27800 Family [ConnectX-5]
```

1.2.2 sui

Bits from sui's dmesg:

```
mlx5_core 0000:b3:00.1: MLX5E: StrdRq(1) RqSz(8) StrdSz(64) RxCqeCmprss(0)
mlx5_ib: Mellanox Connect-IB Infiniband driver v5.0-0
mlx5_core 0000:b3:00.1 enp179s0f1: Link up
```

And ifconfig

```
enp179s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::c058:3aa8:875d:7889  prefixlen 64  scopeid 0x20<link>
        ether 98:03:9b:97:40:17  txqueuelen 1000  (Ethernet)
        RX packets 104783  bytes 35130912 (33.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52384  bytes 8703056 (8.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

And lspci

```
b3:00.0 Ethernet controller: Mellanox Technologies MT27800 Family [ConnectX-5]
b3:00.1 Ethernet controller: Mellanox Technologies MT27800 Family [ConnectX-5]
```

2 Bootstrap

This section is roughly linear account of bootstrapping the system and my understanding of it.

2.1 OS

On both systems I install a few pakages:

```
# yum install emacs git make cmake bison python3 openssl-devel readline \
      libuuid-devel protobuf-devel libtool centos-release-scl-rh \
      devtoolset-3-gcc devtoolset-3-gcc-c++ \
      autoconf automake unzip iperf3
# update-alternatives --install /usr/bin/gcc-4.9 gcc-4.9 \
      /opt/rh/devtoolset-3/root/usr/bin/gcc 10
# update-alternatives --install /usr/bin/g++-4.9 g++-4.9 \
      /opt/rh/devtoolset-3/root/usr/bin/g++ 10
```

For testing connectivity on the 100 Gbps network I open up a range of ports on both ends.

```
# for n in {5200..5209} ; do firewall-cmd  --zone=public --add-port=$n/tcp --permanent; done
# firewall-cmd --reload
```

2.2 NIC

The 100 Gbps NICs are not configured but from `dmesg` and `ifconfig` one can figure out which ones are which and which are linked to the switch. I pick the `10.0.x.y` subnet to use for the 100 Gbps and I pick a convention for IP addresses based on the NIC device name (the `x=1` from the `f1` of the interface name) and the IP address (`y=117` for main IP 192.168.1.117) of the main NIC.

For `dune`:

```
# ip addr add 10.0.1.117/24 dev enp24s0f1 broadcast 10.0.1.255
# ip r add 10.0.1.115 dev enp24s0f1
# ip link set dev enp24s0f1 up mtu 9000
```

For `sui`:

```
# ip addr add 10.0.1.115/24 dev enp179s0f1 broadcast 10.0.1.255
# ip route add 10.0.1.117 dev enp179s0f1
# ip link set dev enp179s0f1 up mtu 9000
```

2.3 Mellanox

The 100 Gbps NICs are Mellanox ConnectX-5. Download some software from `https://www.mellanox.com/page/management_tools` and see also `https://access.redhat.com/articles/3082811`.

```
# cd /root
# tar -xf mft-4.12.0-105-x86_64-rpm.tgz
# cd mft-4.12.0-105-x86_64-rpm
# ./install.sh
# yum install rpm-build kernel kernel-tools kernel-devel
# reboot
# ./install.sh
# mst start
```

Some status queries:

```
# mst status

# yum install mstflint
# mstconfig -d 18:00.0 q
# mstconfig -d 18:00.1 q

# mlxlink -d net-enp24s0f1
```

2.4 iperf3

The `iperf3` program runs a very commonly used network benchmark. It requires one instance to play the role of a server and one the client. With no special tuning, a single `iperf3` client and server can get about 20Gbps on this 100 Gbps network. The tests use ports (5200-5209) opened previous. Here describes running multiple instances
`https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/multi-stream-iper`

```
## on server (.115)
# for port in {5200..5209}; do (iperf3 -s -B 10.0.1.115 -p $port &); done

## on client (.117)
# for port in {5200..5209}; do (iperf3 -c 10.0.1.115 -T s$port -p $port &); done
```

This shows 10 approximately simultaneous tests, each achieving about 10 Gbps.
Following tuning advice from that es.net page.

```
# cpupower frequency-set -g performance
# watch -n 1 grep MHz /proc/cpuinfo
```

A single `iperf3` still gives only 20.9 Gbit/sec. Now try some suggested `sysctl` tuning.

```
# sysctl -a > sysctl-initial.out
# sysctl -psysctl-tune.txt
net.core.rmem_max = 268435456
net.core.wmem_max = 268435456
net.ipv4.tcp_rmem = 4096 87380 134217728
net.ipv4.tcp_wmem = 4096 65536 134217728
net.core.netdev_max_backlog = 250000
net.ipv4.tcp_no_metrics_save = 1
net.ipv4.tcp_congestion_control = htcp
net.ipv4.tcp_mtu_probing = 1
net.core.default_qdisc = fq
```

Iperf3: 19.2 Gbits/sec. Restore prior behavior:

```
# sysctl -psysctl-initial.out
```

Retest `iperf3` and get 20.1 Gbits/sec. Maybe no significant change, especially not compared to the "missing" ~80 Gbps.

2.5 Jumbo Frames

I have heard that setting Jumbo Frames (large MTU) can help throughput. Try the naive thing and set MTU to 9000 on the NICs and test:

```
## on .115
# # ip link set dev enp179s0f1 up mtu 9000
## on .117
# ip link set dev enp24s0f1 up mtu 9000
# ping -s 8972 -M do -c 4 10.0.1.115
```

This fails outright. Probably the switch needs some setting (found below).

2.6 Serial console

Access to some management interface on the switch is needed. Apparently there are several including:

- serial console
- SSH and TELNET
- HTTP ("J-Web")

To get the latter one needs to start on the serial console. Kai and Dimitrios found a way to physically connect this 100 Gbps device over a 9600 baud link to `dune`. I use `minicom` with serial setup 9600 8N1, `/dev/ttyS0`.

Initial login over serial gets root with no password, the system continually sprays messages overy the serial connection and the connection freezes frequently. Frequent killing of `minicom` and reconnecting is needed as is a lot of blind typing.

The Juniper documentation for what comes next is rather bad. The PDF manual is missing lines and the HTML version has typos. After much flailing, below is a reconstruction of what was done.

```
(login)
cli
configure
set system root-authentication plain-text-password
(password)
(password)
commit
delete chassis auto-image-upgrade
commit
set routing-options static route default next-hop 192.168.1.1
set interfaces em0 unit 0 family inet address 192.168.1.127/24
commit
set system services ssh
commit
```

Can now `ssh` as `root@192.168.1.127`. Some info

```
root@qfx5200> show system alarms
2 alarms currently active
Alarm time               Class  Description
2019-10-02 04:08:37 UTC  Minor  JBS For Node locked licenses needed for this device
2019-09-26 03:33:05 UTC  Minor  Rescue configuration is not set

root@qfx5200> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                WH0218510023      QFX5200-32C-32Q
Pseudo CB 0
Routing Engine 0          BUILTIN      BUILTIN           RE-QFX5200-32C-32Q
FPC 0            REV 32   650-059719   WH0218510023      QFX5200-32C-32Q
  CPU                     BUILTIN      BUILTIN           FPC CPU
  PIC 0                   BUILTIN      BUILTIN           32X40G/32X100G-QSFP
    Xcvr 4       REV 01   740-061411   1FCS4431005       QSFP28-100G-AOC-10M
    Xcvr 28      REV 01   740-061411   1FCS4429007       QSFP28-100G-AOC-10M
Power Supply 0   REV 05   740-053352   1GD18191166       JPSU-850W-AC-AFO
Power Supply 1   REV 05   740-053352   1GD18191165       JPSU-850W-AC-AFO
Fan Tray 0                                              QFX5200 Fan Tray 0, Front to Back Airflow - A
```

```
Fan Tray 1                                                    QFX5200 Fan Tray 1, Front to Back Airflow - A
Fan Tray 2                                                    QFX5200 Fan Tray 2, Front to Back Airflow - A
Fan Tray 3                                                    QFX5200 Fan Tray 3, Front to Back Airflow - A
Fan Tray 4                                                    QFX5200 Fan Tray 4, Front to Back Airflow - A

WYixRm-XNEFt-UkSuzR
```

2.7 More iperf3

Can get 28.5 Gbits/sec if the `iperf3` server is pinned to a CPU.

```
## on .117
# tc qdisc add dev enp24s0f1 root fq

## on .115
# tc qdisc add dev enp179s0f1 root fq
```

2.8 More playing on the switch

```
monitor interface traffic
configure
set interfaces et-0/0/4 mtu 9216
set interfaces et-0/0/28 mtu 9216
commit
```

That gets 39.3 Gbps in `iperf3`

2.9 ZeroMQ

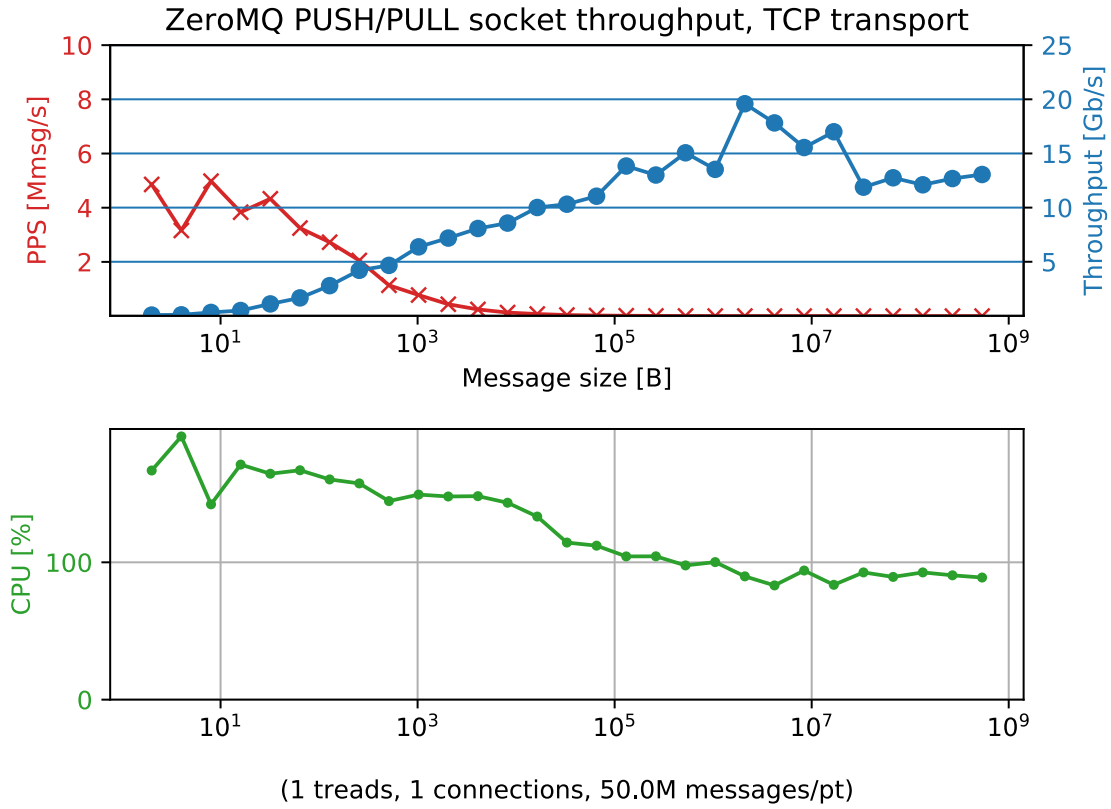
For ZMQ (and PTMP) I have a script (`file://ssh:dune.felix:/home/bviren/dev/bootstrap.sh`) to build from source on `dune` and `rsync` to the same directory on `sui`. It installs tip of the `master` branch of `libzmq`. After building and from inside the `libzmq` source, the ZMQ performance benchmarks can be run like:

```
$ cd dev/libzmq/perf
$ CC=gcc-4.9 CXX=g++-4.9 \
     REMOTE_IP_SSH=10.0.1.115 \
     LOCAL_TEST_ENDPOINT="tcp://10.0.1.117:5200" \
     REMOTE_TEST_ENDPOINT="tcp://10.0.1.115:5200" \
     REMOTE_LIBZMQ_PATH=$(pwd) \
     ./generate_csv.sh
```

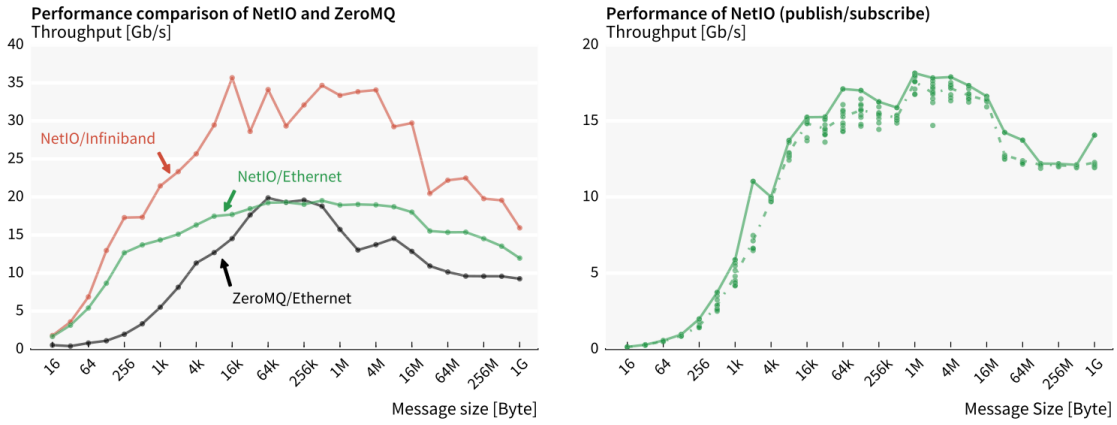
To create the plots from the results, some Python installation is needed:

```
$ python3 -m venv ~/dev/venvs/zmqbm
$ source ~/dev/venvs/zmqbm/bin/activate
$ pip install matplotlib
$ python generate_graphs.py
```

With the default MTU of 1500, the PUSH/PULL throughput reaches 20 Gbps with the largest message size. After the MTU increase, it reaches 25 Gbps. A summary of the results are shown below.



These results (note, it seems jumbo frame MTU was lost in these plots which is why the peak is at 20 Gbps) are roughly consistent with the tests reported on the ZMQ list and on the wiki at: `http://wiki.zeromq.org/results:100gbe-tests-v432` The throughput measurements are consistent with the NetIO comparison with ZeroMQ. The money plots from that summary are reproduced here:



(a) Point-to-Point (b) Publish/Subscribe

Figure 3. Throughput performance of NetIO sockets.

2.10 CPU bottleneck

After almost giving up on why 100 Gbps can not be achieved, it was observed that the `iperf3` client was running at 100% CPU (server at about 75%). This shows the client is single-threaded and CPU bound. And, the previous 10-client/10-server test showed `100 Gbps in aggregate. Repeating with 3 ~iperf` clients and servers still did not saturate the network but did saturate the client CPU. 4 clients allowed an aggregate of 90 Gbps an client CPU usage of about 75%.

Observing the sender end of the ZMQ benchmark in `top` showed it was at 150% CPU. ZMQ uses background threads for I/O, separate from the "main" thread or other application level threads. The hypothesis is that the ZMQ output thread is CPU-bound and that this explains the ZMQ benchmark not saturating the network.

2.11 ZeroMQ benchmark

Investigate how to modify the ZMQ benchmark to use more threads with the goal to saturate the network with a single executable. The zguide suggests one I/O thread per GB/s.

`http://zguide.zeromq.org/page:all#toc31`

Naively, then setting 12 threads will bring magic. However, after modifying the `perf/{local,remote}_thr` to accept an CLI argument to set the number of I/O threads, the result is still a single I/O thread pegged at 100% (as per `top -d1 -H`). Here's the run with 10 I/O threads:

```
[bviren@sui dev]$ ./libzmq/perf/.libs/remote_thr tcp://10.0.1.117:5200 131072 1000000 0 10
using 10 I/O threads

[bviren@dune dev]$ ./libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 131072 1000000 0 10
using 10 I/O threads
message size: 131072 [B]
message count: 1000000
mean throughput: 21952 [msg/s]
mean throughput: 23018.414 [Mb/s]
```

During the run the `local_thr` (app) thread is at 10% and a single I/O thread is at 100%. The other nine are observed at 0%. On the sender, the `remote_thr` (app) thread is essentially at 0% and one I/O thread is at 30-50% and the other nine at 0%. So, sender is not yet bottlenecked even though its only using one thread.

Try pinning to a CPU:

```
# service irqbalance stop
[root@dune ~]# taskset 8 ~bviren/dev/libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 131072 1000000
using 10 I/O threads
message size: 131072 [B]
message count: 1000000
mean throughput: 31926 [msg/s]
mean throughput: 33476.855 [Mb/s]
# service irqbalance start
```

During this run the I/O thread was at 80-90%.

Try even larger message sizes.

```
[bviren@dune dev]$ ./libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 262144 1000000 0 10
using 10 I/O threads
message size: 262144 [B]
message count: 1000000
mean throughput: 12553 [msg/s]
mean throughput: 26325.583 [Mb/s]
```

The next power of 2 doesn't change significantly. Try it also while pinned:

```
[root@dune ~]# taskset 8 ~bviren/dev/libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 524288 1000000
using 10 I/O threads
message size: 524288 [B]
message count: 1000000
mean throughput: 8436 [msg/s]
mean throughput: 35384.354 [Mb/s]
```

2.12 Multiple connections

Doron Somech (zmq guru) responds to my question on `zeromq-dev` saying all that is needed is to call `zmq_connect()` multiple times. He wasn't kidding!

10 threads in both `remote_thr` and `local_thr`, 10 connect() calls in `remote_thr`.

```
[bviren@dune dev]$ ./libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 131072 1000000 0 10
using 10 I/O threads
message size: 131072 [B]
message count: 1000000
mean throughput: 63112 [msg/s]
mean throughput: 66178.185 [Mb/s]
```

10 threads in both `remote_thr` and `local_thr`, 100 connect() calls in `remote_thr`.

```
[bviren@dune dev]$ ./libzmq/perf/.libs/local_thr tcp://10.0.1.117:5200 131072 1000000 0 10
using 10 I/O threads
message size: 131072 [B]
message count: 1000000
mean throughput: 91633 [msg/s]
mean throughput: 96084.376 [Mb/s]
```

In the second case, the 10 threads in $\text{local}_{\text{thr}}$ are using between 50-100% CPU. $\text{remote}_{\text{thr}}$ is much less active with about half threads around 50% and half around 10%.

3 Report

We should write this out in some more organized fashion. It should include systematic look at performance in terms of

- latency
- throughput
- CPU usage

as a function of

- message size
- number of I/O threads
- number of connections

Generation and visualization of these performance benchmarks should ideally be folded back into `libzmq/perf/`.

3.1 Prepare

SSH keeps breaking the connection if idle.

```
# yum install tmux
```

In four local terminals

```
$ tmux new-ses dune-user
$ tmux new-ses dune-root
$ tmux new-ses sui-user
$ tmux new-ses sui-root
```

Develop a Python script to launch the benchmarks and a shell script to iterate over top level parameters.

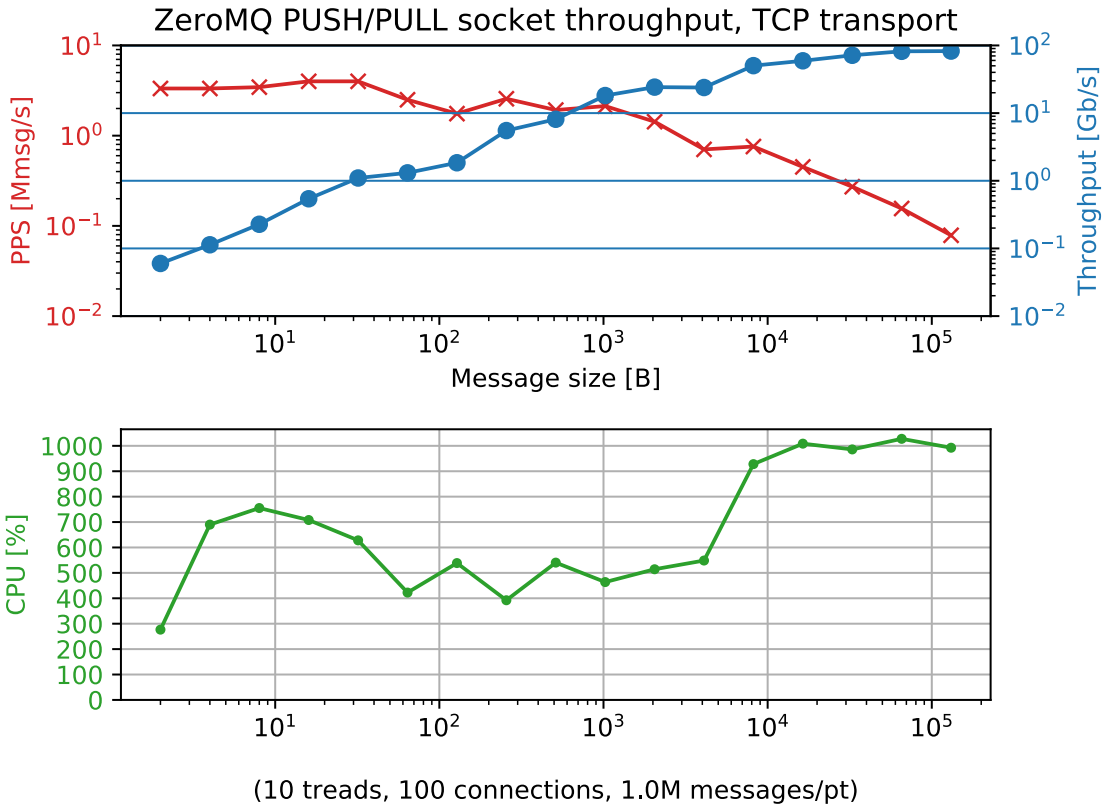
```
$ ./zmqbm.sh
```

Which runs various forms of

```
$ ./zmqbm.py generate --nthreads=10 --nconnections=100 --nmsgs=1000000 -o junk3.json
```

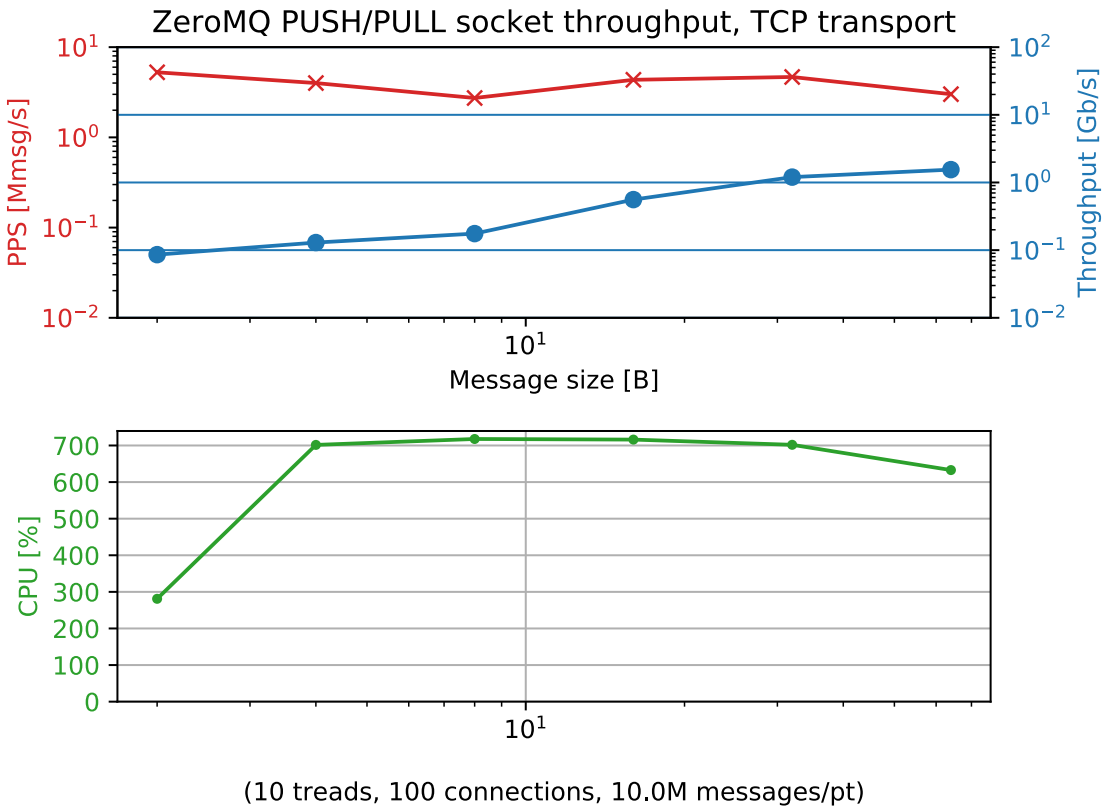
The JSON file can be turned into a plot

```
$ ./zmqbm.py plot-thr -o junk3.pdf junk3.json
```



The %CPU is for the `local_thr` job calculated as `(user+system)/elapsed` with values collected by pre-pending `/usr/bin/time` to this command.

I'm too lazy to include proper uncertainties but running several jobs it's clear that there is a lot of variation in the %CPU for small message. Since these small-message jobs finish in less than a second, I'm guessing "elapsed" is strongly influenced by process startup/terminate times which fairly should not be included for a steady-state. So, I did some special runs with 10M messages for a few of these points and they show a fairly flat 700-800% CPU usage. Example is shown below.

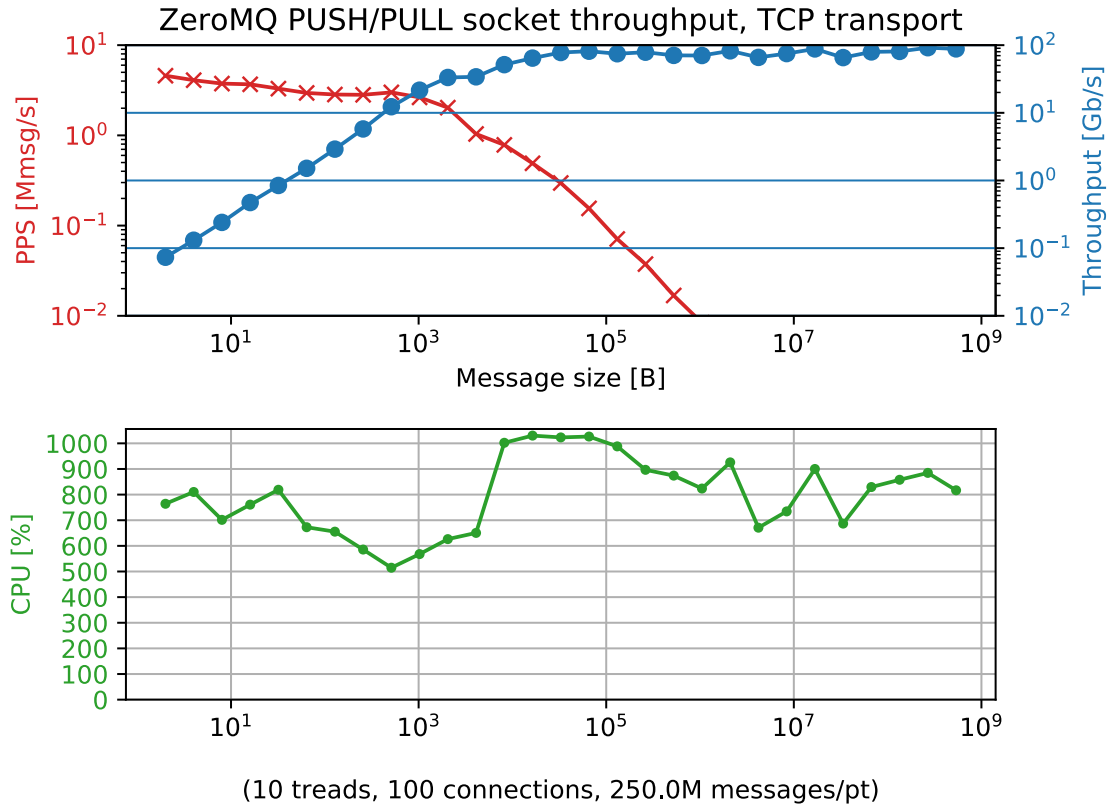


OTOH, the valley of %CPU for mid-size messages is robust. There are also variations in the PPS and throughput across the multiple runs but I would categorize the above example as being fairly representative and the variations not being qualitatively significant.

They test out to 1GB while above tests goes only to 100kB. As messages become bigger doing the tests over a fixed number of messages becomes prohibitive, so change the code to allow to specify a data volume.

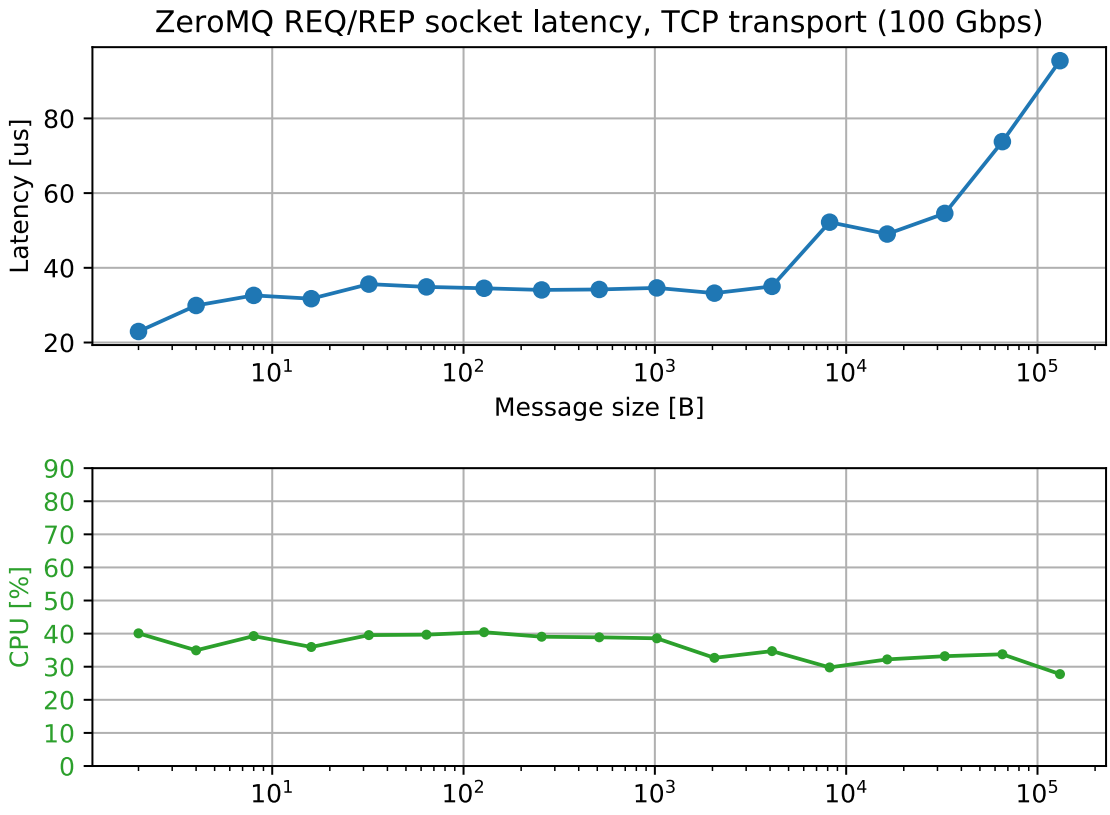
```
do_one () {
    name="zmqbm-vol-10-100-${1}-${2}"
    if [ ! -f $name.json ] ; then
        ./zmqbm.py generate --volume=$3 --nthreads=10 --nconnections=100 \
                    --nminmsglog2=$1 --nmaxmsglog2=$2 -o $name.json
    fi
    ./zmqbm.py plot-thr -o $name.png $name.json
}

do_one 1 5 500M
do_one 5 10 10G
do_one 10 30 100G
```

Result is below. After about 10 kB message size, ZeroMQ can saturate 100 Gbps with 10 threads.



Latency is also checked via REP/REQ. The latency sums the total time to send and receive a given number of messages of the given size and divides by twice that number. The messages is sent through an "echo" type server (`local_lat`) which does no application-level memory allocation. The initiator (`remote_lat`) also reuses the same memory for each message.

The latency results are summarized as a function of message size in the figure below. Latency is flat at about 35 μs until the message approaches the MTU. These results are actually lower than what was reported on the ZMQ wiki.


ZeroMQ REQ/REP socket latency, TCP transport (100 Gbps)
Latency [us]
20
40
60
80
10^1
10^2
10^3
10^4
10^5
Message size [B]
CPU [%]
0
10
20
30
40
50
60
70
80
90
10^1
10^2
10^3
10^4
10^5

3.2 PUB/SUB